

A novel Quorum Protocol

Parul Pandey, Maheshwari Tripathi

1

Abstract—One of the traditional mechanisms used in distributed systems for maintaining the consistency of replicated data is voting. A problem involved in voting mechanisms is the size of the Quorums needed on each access to the data. In this paper, we present a novel and efficient distributed algorithm for managing replicated data. We impose a logical wheel structure on the set of copies of an object. The protocol ensures minimum read quorum size of one, by reading one copy of an object while guaranteeing fault-tolerance of write operations. Wheel structure has a wider application area as it can be imposed in a network with any number of nodes.

Index Terms—Replica-control, distributed database, quorum consensus.

1 INTRODUCTION

IN a distributed database system, data is replicated [17], [14], [8] to achieve fault-tolerance. One of the most important advantages of replication is that it masks and tolerates failures in the network gracefully and increases availability. In particular, the system remains operational and available to the users despite failures. In case of multiple access a problem that must be solved while using replication is how to maintain the copies in a consistent state [7]. To keep logical data consistent, there must exist a control protocol responsible for synchronizing the access. A popular method for maintaining consistency of replicated data is weighted voting [6] which is a generalization of the majority consensus method presented in [16]. In the quorum consensus (QC) [9], [18] algorithm, we assign a non-negative weight [5] to each copy x_A of x . We then define a read threshold RT and write threshold WT for x , such that both $2WT$ and $(RT + WT)$ are greater

than the total weight of all copies of x . A read (or write) quorum of x is any set of copies of x with a weight of at least RT (or WT). For better performance, some logical structure is imposed on the network, and the quorums are chosen under the consideration of such structures. Such logical structures include the tree [3], diamond [4], ring [10], triangular mesh [2], and grid [13] structures. A geometric approach for dealing with logical structures is proposed in [19].

In this paper we propose a novel protocol, which is called *The Wheel Quorum Consensus Protocol* or simply *The Wheel Protocol*, for managing replicated data. In this protocol, the sites in the network are logically organized into a wheel structure. This protocol can be viewed as specialized version of ring and tree protocol. This protocol has an upper hand on both tree and ring protocol, unlike tree and ring protocol it's read quorum size never exceeds one, which is minimum among all. As compared to tree, grid, diamond and mesh protocol, wheel protocol is very flexible in arranging nodes in a network into the logical structure. Any

Maheshwari Tripathi is working with the Computer Science Dept of IET, India

number of nodes can be easily organized into a wheel structure.

The paper is organized as follows. In Section 2 we describe the system model. Section 3 discusses wheel quorum protocols which elaborates the motivation behind it, wheel structure and its quorum construction for read and write.

2 MODEL

A distributed system consists of a set of distinct sites that communicate with each other by sending messages over a communication network. No assumptions are made regarding the speed, connectivity, or reliability of the network. It is assumed that sites are fail-stop [15] and communication links may fail to deliver messages.

Replication of data is achieved by storing copies of the same logical data item at different nodes. Read and write operations can be performed on replicated data. A node needs to obtain permission from a number of copies (quorum) before performing the operation using a control protocol.

In a replicated database, copies of an object may be stored at several sites in the network. Multiple copies of an object must appear as a single logical object to the transaction. This is termed as one-copy equivalence [1] and is enforced by the replica control protocol. The correctness criteria for replicated databases is one-copy serializability [1], which ensures one-copy equivalence and serializable execution of transactions. In order to ensure one-copy equivalence, a replicated object z may be read by reading a read quorum of copies, and it may be written by writing a write quorum of copies. The following restriction is placed on the choice of quorum assignments:

Quorum Intersection Property: For any two operations $o[Z]$ and $\delta[z]$ on an data item x , where at least one of them is a write, the quorums must have a nonempty intersection.

Version numbers or timestamps are used to identify the current copy in a quorum. Each node is logically characterized by few attributes as shown in figure 1. **ID** which is a unique sequential ID. In our discussion, IDs are numbered as 0, 1, 2, 3,... n . **Node Location** is the location where the node is physically residing. In other words this is the address of a node in the network. **HUB** contains the ID of the node in the wheel which is currently acting as hub. In our discussion, ID of the HUB node is 0. **SUC** contains the ID of the successor w_{i+1} , which is the next node in the wheel. **PRED** contains the ID of the predecessor w_{i-1} , which is the previous node in the wheel.

ID	Node Location	HUB	SUC	PRED
----	---------------	-----	-----	------

Fig. 1: Wheel Structure

The election quorum ensures that the HUB's ID is always 0.

3 WHEEL QUORUM PROTOCOL

3.1 Motivation

Tradeoff between the cost for reading, writing, data availability and node fault tolerance is the deciding feature of all existing control protocols for replicated data. For example, the read-one write-all scheme needs only one copy as read quorum, but has the convenience of having a write quorum equal to the total number of copies (thus not tolerating a single node of failure).

The main motivation for our work was to develop a protocol which had a constant

minimum cost for reading, while maintaining an acceptable cost for writing, since we are interested in systems where read operations are much more frequent than write operations.

To achieve this property, a logical wheel structure will be imposed on the set of copies of the object. This structure is used by operations to determine the copies that must be read or written. Figure 2, represents 4 nodes arranged in a wheel structure. Wheel logical structure can be arranged on any number of nodes, whereas other logical structures have constraints with nodes arrangement. We note that this structure is logical, and does not have to correspond to the actual physical structure of the network connecting the sites, storing the copies. This wheel structure is used to motivate the protocol.

3.2 The Wheel Structure

Let $W_n = w_0, w_1, w_2, \dots, w_{n-1}$ be the set of nodes that store copies of a replicated data item. A **wheel**, W_n is a logical structure with n nodes, formed by connecting a single node called HUB to all vertices of an $(n-1)$ cycle. The numerical notation for wheels is used inconsistently in the literature: some authors instead use n to refer to the length of the cycle, so their W_n is the graph we would denote as W_{n+1} . All nodes in the cycle maintain adjacency relationship by maintaining ID's of their successor and predecessor. Each node is defined by attributes ID, Node_Location, HUB, Suc, and Pred as shown in figure 1. Wheel structure is easily imposed on the set of nodes by selecting first node as HUB and adding other nodes as spokes in cycle by defining the successor (Suc(i)), predecessor (Pred(i)) operations and by setting HUB in each spoke. Other operations are GetPermission(i) and rand(1..n).

GetPermission(i), returns TRUE if the node w_i allows access to its own copy of the item. GetPermission(i) returns FALSE when either node w_i refuses access or cannot be contacted due to failure. rand(1..n) selects and returns random number from 1 to n, where n is the number of nodes in wheel. This random number represents ID of selected node.

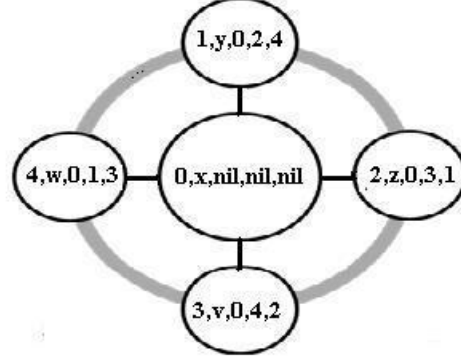


Fig. 2: Wheel Structure

One of the restrictions imposed by the suggested implementation for collecting read quorums is that the reads are directed to a specific copy: the HUB. This has the advantage that if the HUB is up, read operations accesses a single copy. Read locality may, however, be sacrificed and the HUB may become a bottleneck. To solve this problem, it is desirable to gather a quorum of several relatively-local copies rather than one very remote HUB copy. This approach could also be used for organizing the wheel structure of the copies. For example, consider a network composed of two relatively distant segments: the HUB could be placed in one of the segment and the other nodes of the wheel in the other segment. In such an organization, transactions executing in a particular segment will use the quorum which is less expensive. If the HUB is in the transaction's network segment, the HUB will be accessed. Oth-

erwise, the transaction will access any two adjacent nodes of the wheel. The functions depicting Read and Write quorum should be appropriately modified to enforce this policy. Whereas, one policy of election quorum is already suggested in this paper to avoid the problem of HUB bottleneck.

3.3 The Wheel Protocol

In this protocol, all copies of a replicated data item are organized into a wheel structure. Specific algorithms are used for read and write quorums construction. There is one election algorithm for electing new HUB in case of failure of HUB or in case load threshold exceeds its limit. These algorithms use the adjacency information to guarantee quorum intersection, and to maintain the quorum sizes small. There are three type of quorums, Read, Write, and Election quorum.

Read Quorum is formed by getting access permission from HUB.

Write quorum is obtained by getting access permission from HUB and half of alternating nodes in the cycle, thus requiring the majority of the total number of copies. As an example, consider a replicated data item with six copies arranged in a wheel structure as shown in figure 3. Eligible read quorum is 0 (i.e. HUB) and sets eligible for write quorum are : $\{0,1,3,5\}$, $\{0,1,2,4\}$, $\{0,3,5,2\}$, $\{0,4,1,3\}$ and $\{0,5,2,4\}$.

Notice that eligible quorums are coteries, satisfying the minimality and intersection properties¹.

Election quorum is called in two situations

- 1) When HUB crosses its load threshold
- 2) When HUB is unavailable

1. The fact that the quorums are distinct and have the same size shows that they satisfy the minimality property: the intersection property will be shown later, when providing the protocol correctness

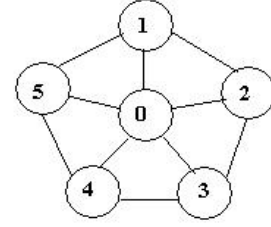


Fig. 3: 6 copies organized into a logical structure

In both the above cases, the node initiating election quorum algorithm, selects randomly any 2 adjacent nodes, checks their version and makes the latest one the HUB by changing the location address between the old HUB and the newly elected one. This logically swaps the location of the two nodes. Other nodes are unaffected as they identify HUB by its ID, which is 0. Only the node_location is changed.

Advantages of this Election Quorum are-

- 1) HUB is never overloaded, as it gets swapped with a latest node whenever load _ threshold crosses its limit.
- 2) Improved load distribution. Assuming that each node in cycle has the equal probability of being selected as a new HUB, no node will be working as HUB for a longer time.
- 3) Constant minimum possible Read Quorum size of one. As, even if HUB is failed , it will be replaced with a new HUB. Thus ensuring that a request always reads data from HUB.

Without using election quorum, in the failure of HUB, Read Quorum can be achieved by accessing any 2 adjacent nodes in the cycle, which is double the cost of doing it with HUB. Our system has more number of reads as compared to write , so reads will keep on costing double till HUB recovers. All this can be avoided by using election quorum and electing new HUB. This

way, present as well as subsequent reads can be satisfied by reading only HUB.

3.3.1 Quorum Construction

There are three algorithms for the wheel protocol. Algorithm 1, 2, 3 for read, write and election quorum respectively.

Algorithm 1 defines read quorum construction. This algorithm returns the HUB as the read quorum. In case of a HUB failure, the new HUB is elected by invoking the ElectionQuorum Protocol, which uses a random node in cycle.

Algorithm 1 Read Quorum(i)

```

if Empty(Wheel) then
    Return(nil)
else if GetPermission(HUB) is False then
    r= rand(1 .. n)
    Get ElectionQuorum(r)
    Return(HUB)
else
    Return(HUB)
end if

```

Algorithm 2 is to find write quorum. This protocol collects majority of nodes forming quorum between nodes in cycle of wheel in list, Quorum_list[]. This Quorum_list[] along with HUB makes write quorum. Protocol tries to form write quorum with current_node by traversing the cycle until, either a quorum is obtained or all copies have been examined (in which case quorum was not obtained and the request for writing is refused). In case of HUB failure Election Quorum elects a new HUB.

In case of HUB failure, Election Quorum (Algorithm 3) elects a new HUB. This protocol can be called in two conditions. First when the HUB has failed or second whenever HUB exceeds it's load

Algorithm 2 Write Quorum(i)

– *Main routine*

```

1: nodes_covered=0
2: current_node = i
3: if GetPermission(HUB) is False then
4:   n= random(cycle nodes)
5:   Get ElectionQuorum(n)
6:   GetPermission(HUB)
7: end if
8: if current_node is HUB then
9:   current_node = rand(1..n)
10: end if
11: while Empty QuorumList[] and
    nodes_covered < n do
12:   Quorum_list[]=
    Check(current_node)
13:   current_node=Suc(current_node)
14:   nodes_covered++
15: end while
16: Return(HUB  $\cup$  QuorumList[])

```

– *Check(i)*

```

1: Quorum_list[] = null
2: fail= nodes_checked=0
3: while Fail  $\neq$  1 and nodes_checked <
     $\lfloor n/2 \rfloor$  do
4:   if GetPermission(i) then
5:     Quorum_list.add(i)
6:     i=Suc(Suc(i))
7:     nodes_checked++
8:   else
9:     Fail=1
10:  end if
11: end while
12: if Fail then
13:   Quorum_list.flushall()
14:   return(Quorum_list[])
15: else
16:   return(Quorum_list[])
17: end if

```

threshold. Election quorum selects two adjacent nodes(using successor function), selects the node with latest value and makes it the HUB.

Algorithm 3 Election Quorum(i)

```

1: current_node=i
2: Quorum=0
3: nodes_done=0
4: if current_node is HUB then
5:   current_node=rand(1..n-1)
6: end if
7: while Quorum is Empty or
   nodes_done < n do
8:   if current_node is accessible then
9:     if SUC(current_node) is accessible
       then
10:      Latest_node=Node_Location
        with most recent value
11:      Swap Node_Location of HUB
        and Latest_node
12:      Quorum=Latest_node
13:    else
14:      current_node=Suc(Suc(current_node))
15:      nodes_done=nodes_done + 2
16:    end if
17:  else
18:    current_node=Suc(current_node)
19:    nodes_done=nodes_done+1
20:  end if
21: end while

```

3.3.2 Quorum Size

An outstanding feature of Wheel quorum is its minimum read quorum size which is always one. This is achieved by reading only HUB. HUB is always made available even if existing one is failed by election quorum. This is minimum read quorum size achieved by any algorithm.

Write quorum size is $\lceil (n-1)/2 \rceil + 1$, including the HUB.

3.3.3 Proof of Correctness and Non-equivalence with Vote Assignment

To show protocol correctness, it must be shown that no two conflicting operations are permitted to occur at the same time. Following theorems prove correctness of our algorithms.

Theorem 3.3.1. *In a wheel of size n , the Wheel Protocol guarantees a non-empty intersection between any read and write quorums.*

Proof: The proof follows from read and write quorum construction algorithms. The read quorum is formed by only HUB in the wheel and write quorum selects HUB and alternate nodes from $(n-1)$ nodes of cycle. Both of them will definitely contain HUB and thus ensures non-empty intersection between any read and write quorums. \square

Theorem 3.3.2. *In a wheel of size n , the Wheel Protocol guarantees that there is a non empty intersection between any write quorums.*

Proof: It follows from the fact that write quorum is formed by majority of copies $\lceil (n-1)/2 \rceil + 1$ which includes HUB in the wheel. Since, each write quorum must include HUB, it is guaranteed that the intersection between any write quorums is non-empty. \square

An interesting property of wheel protocol is that the coterie it generates cannot be generated by any vote assignment in the voting protocol[6].

Theorem 3.3.3. *There is no vote assignment equivalent to the wheel protocol.*

Proof: By contradiction. Consider the wheel in figure 2. Let v_0, v_1, \dots, v_5 be the vote assigned to the six copies and V_i be the total number of votes. Consider, the two eligible write quorum sets of copies $\{0, 1, 3, 4\}$ and $\{0, 2, 4, 5\}$, two other sets that are not eligible quorums $\{0, 2, 3, 4\}$ and $\{0, 1, 4$

, 5}. For a vote assignment to be equivalent to the wheel protocol the following must hold

$$v_0 + v_1 + v_3 + v_4 > \frac{V_i}{2} \quad (1)$$

$$v_0 + v_2 + v_4 + v_5 > \frac{V_i}{2} \quad (2)$$

$$v_0 + v_2 + v_3 + v_4 < \frac{V_i}{2} \quad (3)$$

$$v_0 + v_1 + v_4 + v_5 < \frac{V_i}{2} \quad (4)$$

For (1) , (2) there is a quorum and (3), (4) there is no quorum. Solving (1) and (3) we conclude that $v_1 > v_2$. Solving (2) and (4) we conclude that $v_2 > v_1$, which is a contradiction. Therefore, there is no vote assignment (using positive integers) that satisfies both condition, and the theorem follows. \square

Different logical structures have been exploited in [11] and message overhead analysis of wheel is done in [12]

REFERENCES

- [1] P. A. Bernstein and N. Goodman. A proof technique for concurrency control and recovery algorithms for replicated databases. *Distributed Computing, Springer-Verlag*, 2(1):32-44, January 1987.
- [2] Yao-Jen Chang. A triangular-mesh-based approach to fault-tolerant distributed mutual exclusion. Master's thesis, National Sun Yat-sen University, June, 1995.
- [3] Amr E. Abbadi Divyakant Agrawal. The tree quorum protocol: An efficient approach for managing replicated data. *Proceedings of the 16th International Conference on Very Large Data Bases (1990)*, pages pp. 243–254., 90..
- [4] Ada Wai-Chee Fu, Yat Sheung Wong, and Man Hon Wong. Diamond quorum consensus for high capacity and efficiency in a replicated database system. *Distrib. Parallel Databases*, 8:471–492, October 2000.
- [5] Hector Garcia-Molina and Daniel Barbara. How to assign votes in a distributed system. *J. ACM*, 32:841–860, October 1985.
- [6] H. Gifford. Weighted voting for replicated data. *in Proceedings of 7th Symposium on operating Systems, ACM*, pages pp 150–162, 1979.
- [7] Jim Gray, Pat Helland, Patrick O'Neil, and Dennis Shasha. The dangers of replication and a solution. In *SIGMOD '96: Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, volume 25, pages 173–182, New York, NY, USA, June 1996. ACM.
- [8] Yi Lin, Bettina Kemme, Marta Patiño Martánez, and Ricardo Jimáñez-Peris. *Consistent Data Replication: Is It Feasible in WANS?* 2005.
- [9] M. L. Liu, D. Agrawal, and El A. Abbadi. Abbadi. On the implementation of the quorum consensus protocol. In *In Proc. Parallel and Distributed Computing Systems*, 1995.
- [10] Nabor C. Mendona and Ricardo O. Anido. The hierarchical ring protocol: An efficient scheme for reading replicated data. Technical Report DCC-93-02, Department of Computer Science, University of Campinas, February 1993. In English, 30 pages.
- [11] M.Tripathi Parul Pandey. Exploiting logical structures to reduce quorum sizes of replicated databases. *Advanced Computing : An International Journal* (2012),, 3:99–104, January 2012.
- [12] M. Tripathi P.Pandey. Message overhead analysis of quorum protocol. *in Proceedings of International Conference on Advances in Computing*, 174:pp 237–245, 2012.
- [13] M. H. Ammar S. Y. Cheung and M. Ahamad. The grid protocol: A high performance scheme for maintaining replicated data. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 4, No. 6:pp. 582–592, Dec. 1992.
- [14] Yasushi Saito and Marc Shapiro. Optimistic replication. *ACM Comput. Surv.*, 37(1):42–81, March 2005.
- [15] Richard D. Schlichting and Fred B. Schneider. Fail-Stop Processors: An Approach to Designing Fault-Tolerant Computing Systems. *Computer Systems*, 1(3):222–238, 1983.
- [16] Robert H. Thomas. A Majority consensus approach to concurrency control for multiple copy databases. *ACM Trans. Database Syst.*, 4(2):180–209, June 1979.
- [17] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso. Understanding replication in databases and distributed systems. In *In Proceedings of 20th International Conference on Distributed Computing Systems (ICDCS2000)*, pages 264–274, 2000.
- [18] Avishai Wool. Quorum systems in replicated databases: science or fiction. *Bull. IEEE Technical Committee on Data Engineering*, 21:3–11, 1998.
- [19] Y.C.Kuo and S.T. Huang. A geometric approach for constructing coterie and k-coterie. *IEEE Transaction Parallel and Distributed Systems*, 8(4):402–411, April 1997.